Universidad Politécnica de Madrid

Facultad de Informática

Trabajo fin de grado presentado para obtener el grado de

Graduado en Ingeniería Informática

Análisis de prestaciones y eficiencia de tablas de cobertura para CBR (Content-Based Routing)

por

He, Lili R090203

Tutor

Jiménez, Miguel

Madrid - España

Junio de 2013

Análisis de prestaciones y eficiencia de tablas de cobertura para CBR (Content-Based Routing)

He, Lili

9 de junio de 2013

Agradecimientos

Acknowledge my maternal grandmother who is always staring me from heaven y those persons who have helped me. In particular, special thanks to Ramon Galindo who have checked my disorganized texts of Spanish, and Sergio Vavassori who is my superior, helped me to solve those annoying problems and verified all results of those experiments.

Reconozco a mi abuela que siempre está mirándome desde el cielo y a la gente que me ha ayudado. Especialmente agradezco a Ramon Galindo quien ha corregido mi español desorganizado y Sergio Vavassori, mi superior, quien me ha ayudado a solucionar los problemas difíciles y ha verificado todos los resultados con un enfoque científico riguroso.

Resumen

Hoy en día, los sistemas middleware de publicar-suscribir con la filtración de mensajes basado en contenido tiende a ser popularizado, y un sistema como éste requiere codificar su mensaje a la combinación de varios elementos que se encuentran en los conjuntos no-interseccionados. Varios predicados posibles en los dominios de esos conjuntos forman un filtro, y el núcleo de algoritmo filtrado es seleccionar filtros adaptados tan pronto como sea posible. Sin embargo, el conjunto, que está formado por los filtros, contiene la extremadamente fuerte indeterminación y distensibilidad, lo que restringe el algoritmo filtrado.

Por la resolución de la distensibilidad, se estudió la característica del conjunto de filtros en álgebra, y sabía que es un retículo específico. Por lo tanto, se intenta usar el carácter, el cual los retículos forman un conjunto parcialmente ordenado (o poset, del inglés partially ordered set) con límites, para reducir el tamaño de conjunto de filtros (compresión equivalente). Por estas razones, es necesario implementar un contenedor abstracto de retículo, y evaluar su desempeño tanto en la teoría, como en la práctica, para la solución de la distensibilidad del conjunto de filtros.

Retículo (Lattice) es una estructura importante de Álgebra Abstracta, comúnmente se utiliza para resolver el problema teórico, y apenas de ser un contenedor abstracto en la ciencia de software, como resultado de su implementación compleja que proviene de su trivialidad en álgebra. Y por eso se hace difícil mi trabajo.

Con el fin de evitar la teoría compleja del sistema práctico, simplemente introduce su núcleo algoritmo, el algoritmo de conteo, y esto llevó a cabo con el problema - la distensibilidad del conjunto de filtros. A continuación, se investigó la solución posible con retículos en la teoría, y se obtuvo el diseño de la implementación, normas para las pruebas xUnit y parámetros para la evaluación. Por último, señalamos el entorno, el resultado, el análisis y la conclusión de la prueba de rendimiento.

Summary

Nowadays, publish-subscribe middleware systems with content-based message filtering become popular, and a system like this require to encode its message to the combination of various elements that are in the non-intersection sets. Several possible predicates in these dominions of sets format a filter, and the core of the filtering-algorithm is select matched filters as quickly as possible. But the filtering-set, which is formatted for filters, contains extremely strong indeterminacy and distensibility, thus restrict the filtering algorithm.

For solving the distensibility, we studied the Algebra characteristic of the filtering-set, and knew that it is a specific Lattice. So we intent to use the character, which Lattice is a coverage and bounded partially ordered set (or poset), for reducing the size of filtering-set (equivalent compression). For those reasons, we have to implement a Lattice abstract container, and evaluate its performance in theory and practice for solving the distensibility of filtering-set.

As we know, Lattice is an important structure of Abstract Algebra, commonly is used for solving theoretical problem, and hardly ever for being an abstract container in Software Science, as a result of its complex implementation which comes from its triviality in Algebra. And that makes difficulty to my work.

In order to avoid the complex theory of the practical system, we just introduce its core algorithm (Counting Algorithm), and with this led out the problem (the distensibility of the filtering-set). Then we investigated the possible solution with lattice in theory, and obtained the design of implementation, testing rules for unit testing and parameters for the evaluating test. Finally, we point out the environment, result, analysis and conclusion of the performance test.

Índice general

Índice de figuras

Capítulo 1

Introducción y planteamiento del problema

1.1. Introducción

Con la popularidad de los sistemas *middleware*[1] de *publicador y subscriptor*[2], el estudio y la mejora del algoritmo de *enrutamiento*[3] de los mensajes para esos sistemas se convierte en importante. La *CoNWeTLab*, donde estoy trabajando, ha desarrollado un *sistema middleware basado en el contenido*[4] que utiliza el *algoritmo de conteo*[5] como su núcleo, y sobre esta base para mejorar y optimizar. La optimización del *algoritmo de conteo* se concentra en dos aspectos, optimizar su funcionamiento o reducir el número de objetos de su búsqueda, y este artículo se centra en el segundo tema. El *algoritmo de conteo* tiene que realizar una búsqueda sobre un conjunto especial (el conjunto de filtros), debido a eso, si pudiera encontrar una compresión equivalente de este conjunto, el tiempo de la búsqueda se reduciria, y mejoraria el sistema.

Antecedente. Mis compañeros han desarrollado *SilboPS*, un *sistema middleware de publicador y subscriptor* con el *algoritmo de conteo*. Para mejorarlo, se hicieron algunas investigaciones, e introdujeron la cobertura al sistema. Así que todo mi trabajo es mejorar y optimizar el sistema *SilboPS* con este término, y todas las evaluaciones están bajo este sistema.

Problema. Como se menciona en el artículo del *algoritmo de conteo*, la naturaleza del problema es encontrar todos los filtros adaptados tan pronto como sea posible. Sin embargo, cuando se introduce el concepto de cobertura, tenemos que darnos cuenta que, el mantenimiento de la cobertura causa la pérdida del rendimiento del sistema. Si la mejora de *enrutamiento*, que viene de la cobertura, es menor que la pérdida del rendimiento, entonces la cobertura no ha mejorado el sistema total. Y viceversa mejora el sistema.

Objetivo. Mi trabajo es que determinar cuando introduzca la cobertura al sistema, si realmente mejora la eficiencia del sistema. Para eso, tenemos que investigar la relación de cobertura, y luego desarrollar una estructura de datos para la cobertura, finalmente evaluar con el entorno que ya está implementado.

1.2. Limitaciones de la implementación

La implementación ocupa una parte importante del proyecto, dado que es el requisito previo de la evaluación, y las dificultades del desarrollo se convierte en las siguientes dos partes.

1.2.1. Modificaciones sobre el sistema implementado

Es necesario que modifique las interfaces del sistema implementado para soportar el nuevo algoritmo, pero esas modificaciones posiblemente causarán errores al sistema implementado.

1.2.2. Exactitud de la implementación

Como he mencionado en el resumen, el retículo es una estructura bastante genérica en Álgebra, y eso complica la exactitud de la implementación. Es decir, que asegurar los funcionamientos de la tabla es un problema difícil de solucionar.

1.3. Evaluación de la cobertura

El objetivo final es evaluar el impacto de la cobertura sobre el sistema *SilboPS*, y se divide en dos propósitos generales.

1.3.1. Selección de las tablas de cobertura

Seleccionar la mejor implementación de la estructura de datos. Para llevar esto a cabo es necesario crear una nueva implementación con el algoritmo mejorado y evaluarla frente a la ya existente.

1.3.2. Evaluación sobre el impacto al sistema general

Evaluar el impacto sobre el sistema general y obtener la ganancia total. Para conseguir esto, habrá que desarrollar un entorno genérico parametrizado para realizar la evaluación.

1.4. Proceso del trabajo

La figura 1.1 muestra el proceso del trabajo.

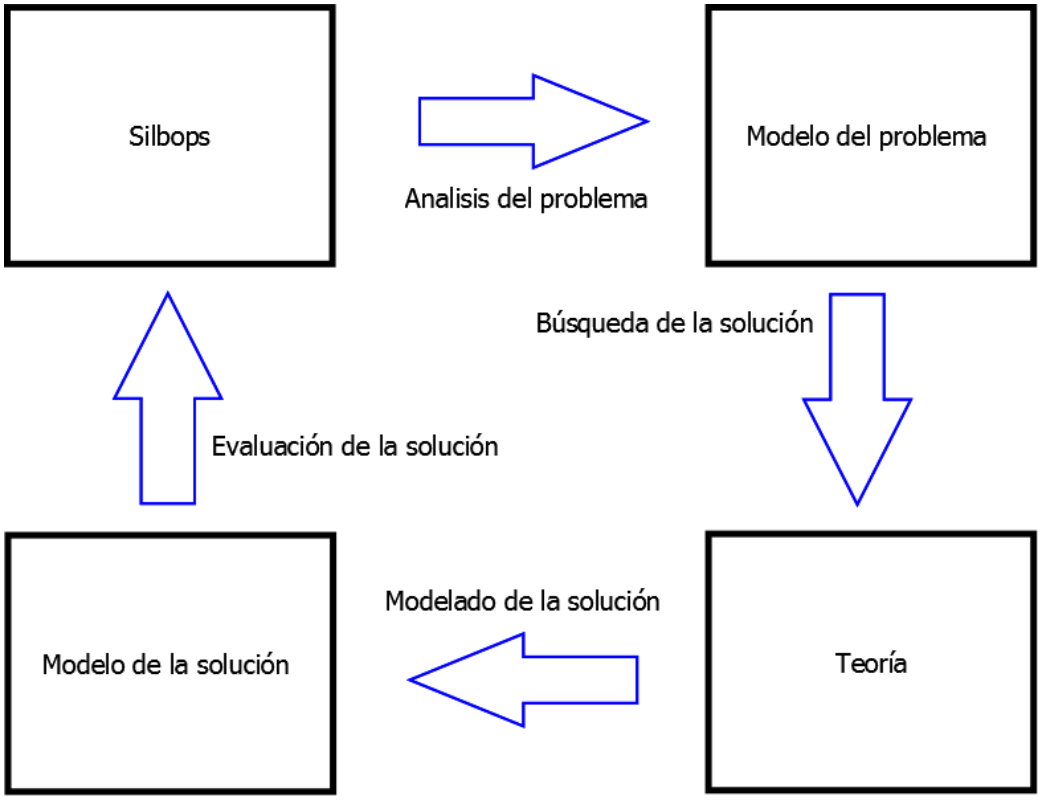

Figura 1.1: El proceso del trabajo

1.5. Esquema del contenido

La figura 1.2 muestra el esquema del contenido del este documento, y las relaciones entre los temas.

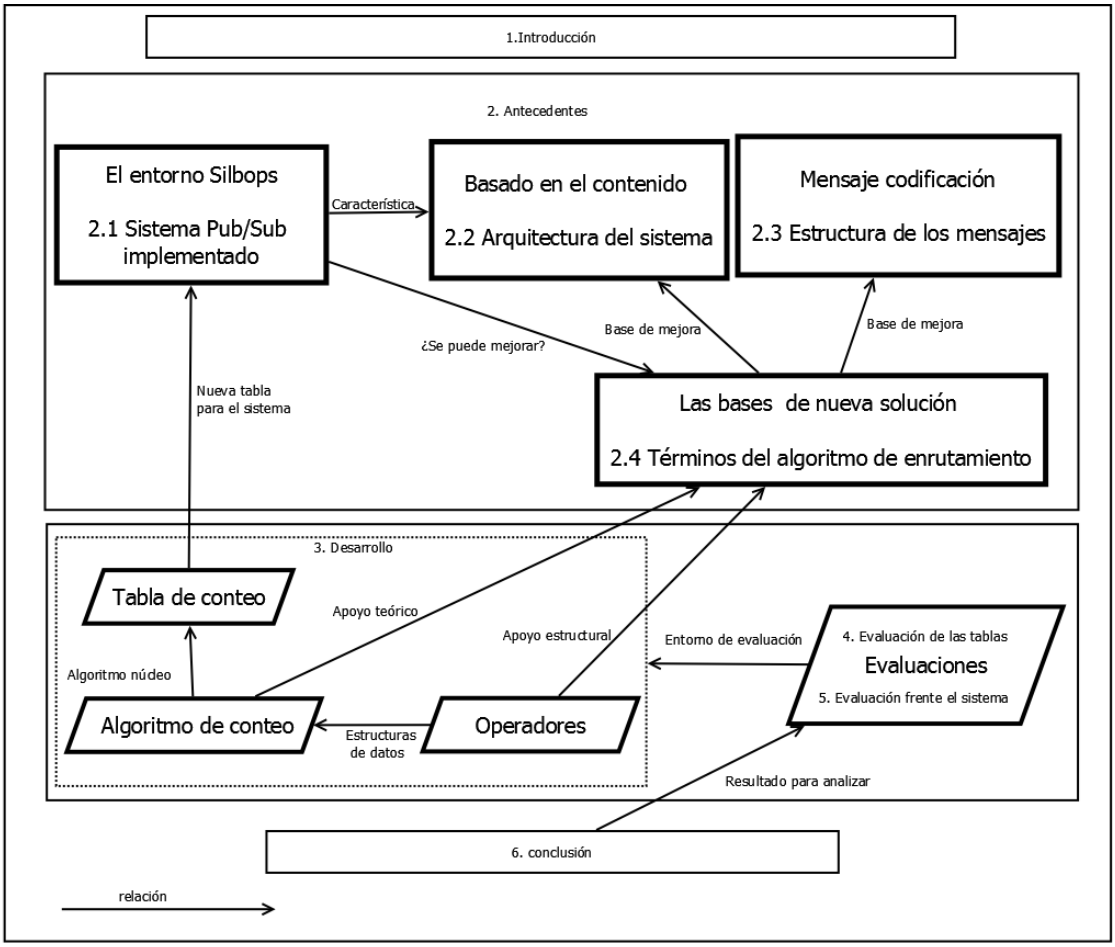

Figura 1.2: El esquema del contenido

Capítulo 2

Antecedentes

Este capítulo va a presentar los conceptos del *sistema publicador y subscriptor (Sistema Pub/Sub)*[6] utilizando el sistema que ya está implementado. Después especificamos su arquitectura y su *algoritmo de enrutamiento*[7] de los mensajes, cuales son los dos factores más importantes para la eficiencia del sistema. A continuación, discutimos *el algoritmo de conteo* y la cobertura de las suscripciones, las cuales forman una base de nuestra solución. Al final, indicamos como mejora *el algoritmo de enrutamiento* usando la cobertura. Por razón de que todo eso es el antecedente y la premisa del trabajo realizado, y con el fin de que comprendáis el alcance, la razón y el límite de nuestro trabajo.

2.1. Sistema publicador y subscriptor implementado

El *SilboPS*, que es el sistema implementado, es un sistema *middlleware*, y sus factores principales son el *publicador y subscriptor mensaje patrón*[8], la codificación de los mensajes y el *algoritmo de enrutamiento*. La figura 2.1 muestra la estructura del sistema. Ahora se muestran los conceptos básicos de nuestro sistema:

1. Nodos de la topología

   - **Publicador**

     El nodo externo, proporciona las informaciones. Puede hacer las acciones como publicar, anunciar, conectar y desconectar.

   - **subscriptor**

     El nodo externo, requiere la información. Puede hacer las acciones como subscribir, conectar y desconectar.

   - ***Broker***[9]

     El nodo intermedio, el cual tiene la responsabilidad de llevar los mensajes a sus destinos. Puede hacer las acciones como reenviar(mensaje), aceptar(conexión) y entregar(mensaje). En el sistema publicador y subscriptor, el *broker* no reenvía los mensajes según la información de origen y destino que ellos poseen, sino según las características

que los mensajes tienen. Es decir, el factor de reenvío de los mensajes fundamentalmente depende de los mensajes ellos mismos, y no depende del origen de los mensajes. En general, el *broker* puede dirigir los mensajes según sus tipos, sus temas, o sus contenidos.

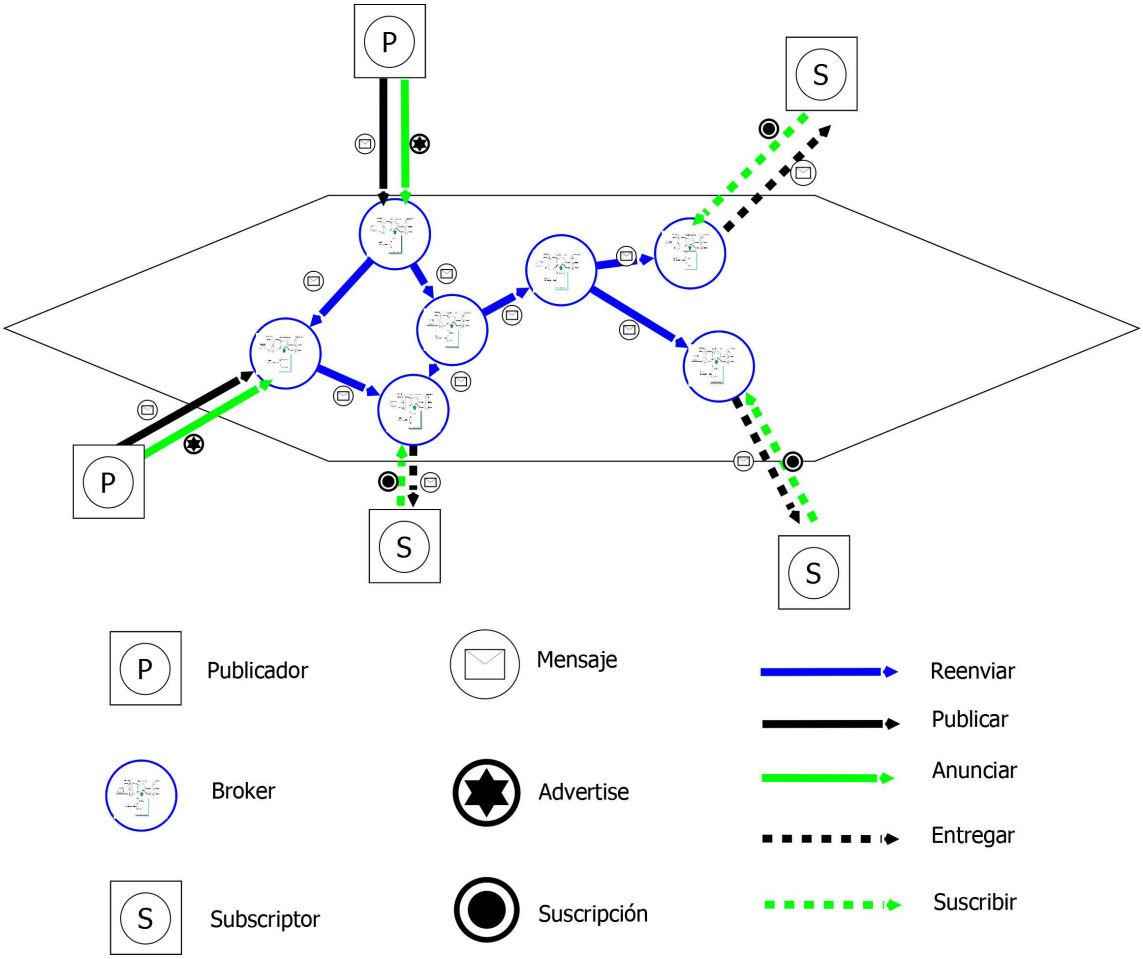

Figura 2.1: El sistema *SilboPS*

2. Mensajes a través del sistema

   - ***Advertise***[10]

     Mensaje que proporciona el publicador para indicar lo que publicará en el futuro.

   - **Suscripción**

     Mensaje que describe la demanda del subscriptor. En general, se representa con un conjunto de restricciones, que indica qué mensajes se desean recibir.

   - **Notificación**

     Mensajes que publican los publicadores.

3. Acciones del nodo

   - ***Reenviar***[11]
     Llevar los mensajes a sus destinos.
   - **Publicar**
     Publicar las notificaciones.
   - **Entregar**
     Entregar las notificaciones a los subscriptores.
   - **Suscribir**
     Suscribir la demanda de las notificaciones.
   - **Anunciar**
     Anunciar la descripción del futuro mensaje.

Los publicadores anuncian sus notificaciones que van a publicar en el futuro, y cuando los subscriptores reciben *advertises* que han publicado, envían una suscripción al sistema en caso si quieres recibir algo sobre lo que van a publicar. Con las suscripciones, el *broker* configura dinámicamente las condiciones de reenvío (tabla de *enrutamiento*), seleccionará y reenviará los mensajes a los subscriptores.

El sistema permite que los nodos se conecten y desconecten dinámicamente sin reiniciar el sistema, además reenvía los mensajes a su destino según la demanda de los subscriptores y el contenido del mensaje. Con lo cual, esos funcionamientos requieren una arquitectura de software eficaz para apoyarlos. Enseguida, describiré la arquitectura del *SilboPS*.

2.2. Arquitectura del sistema

Se utiliza *el patrón publicador - subscriptor para el reenvio de mensajes* en la arquitectura del sistema, y sabemos que un sistema así se puede clasificar en tres clases, que son: *el sistema basado en el tema*[12]*, el sistema basado en el contenido*[13] y *el sistema basado en el tipo*[14], según la filtración de los mensajes que ha utilizado el sistema. Es decir que el sistema reenvía los mensajes según lo que está basado. En nuestro caso, el sistema está implementado como un *sistema basado en el contenido* y reenvía los mensajes según sus contenidos (*Mensaje filtrado basado en el contenido*[15]). Los conceptos fundamentales de un sistema como éste son:

- ***Mensaje filtrado***

  El proceso de la selección y el reenvío de los mensajes. En nuestro caso, se filtra el mensaje según su contenido. La figura 2.2 muestra *el proceso de la filtración*[16] de nuestro sistema.

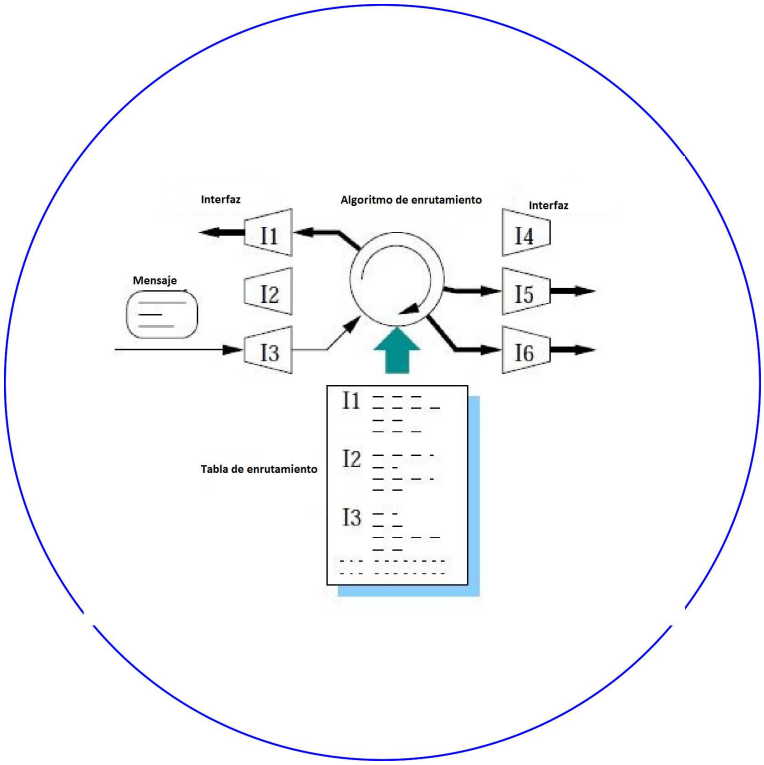

Figura 2.2: El mensaje filtrado con broker

Como vemos, a nivel software, las conexiones están definidas como interfaces, las demandas(suscripciones) que vienen de una conexión se guardan en la tabla de *enrutamiento*, y un algoritmo hace una búsqueda sobre la tabla, en nuestro caso es *el algoritmo de conteo*, y selecciona las interfaces para enviar el mensaje.

- **Topología**

  Como vemos en la figura 2.1, la topología de la red está distribuida en varios *broker*s. Los *brokers* son los nodos que hacen el *enrutamiento* de los mensajes.

- **Ventajas**

  Las dos grandes ventajas que ofrece esta arquitectura son:

  - **Acoplamiento débil**

    Permite la configuración dinámica de los nodos. Los publicadores y subscriptores pueden conectar al sistema en cualquier lugar y cualquier tiempo, tanto como en la desconexión, por eso no existe la dependencia entre los publicadores y subscriptores.

  - **Escalabilidad**

    Con esta característica, el sistema adapta a distintas escalas de mensaje, con lo cual, podemos conseguir que el sistema funcione como un sistema *middleware* y adapta una cantidad enorme de mensajes.

Los factores decisivos de nuestro sistema, o bién de un *sistema publicador y subscriptor basado en el contenido*[17], son: la estructura de los mensajes y su algoritmo del reenvío de mensaje basado en esa estructura, y están descritos en los dos siguientes apartados.

2.3. Estructura de los mensajes

1. Las notificaciones están codificadas como un conjunto de atributos, y cada atributo está definido con tres términos: el nombre, tipo y valor. Como vemos:

$$\text{Atributo} \begin{cases} Nombre & string \\ Tipo & \{\text{“}Integer\text{”}, \text{“}Float\text{”}, \text{“}String\text{”}\} \\ Valor & \begin{cases} Integer\,\{... \quad -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad ...\} \\ Float\,\{... \quad -0{,}2 \quad -0{,}11 \quad 0 \quad 1{,}0 \quad 3{,}0 \quad ...\} \\ String\,\{\text{“}españa\text{”} \quad , \text{“}china\text{”} \quad , \text{“}italia\text{”} \quad , \text{“}pepe\text{”} \quad ...\} \end{cases} \end{cases}$$

   El nombre es una cadena simple de letras que describe el nombre del atributo y el tipo describe el dominio del valor, y el valor del atributo puede ser texto o número. Por ejemplo, un billete de avión de China a España, con precio 600,13 euros y el trayecto dura 18 horas puede codificar como:

$$\text{El billete} : \begin{array}{lll} \text{Nombre} & \text{Tipo} & \text{Valor} \\ \begin{cases} \text{“}Destino\text{”} \\ \text{“}Origen\text{”} \\ \text{“}Precio\text{”} \\ \text{“}Duración\text{”} \end{cases} & \begin{cases} \text{“}String\text{”} \\ \text{“}String\text{”} \\ \text{“}Float\text{”} \\ \text{“}Integer\text{“} \end{cases} & \begin{cases} \text{“}España\text{”} \\ \text{“}China\text{”} \\ \text{“600,13”} \\ \text{”18”} \end{cases} \end{array}$$

2. Las *advertises* del publicador, están codificadas como:

$$\textit{Advertise}: \begin{cases} Nombre & string \\ Tipo & \{\text{“}Integer\text{”}, \text{“}Float\text{”}, \text{“}String\text{”}\} \end{cases}$$

   Una *advertise* de la agencia de billetes de avión puede estar como:

$$\text{La } \textit{advertise} \text{ de billete} : \begin{cases} \text{“}Destino\text{”} & \text{“}String\text{”} \\ \text{“}Origen\text{”} & \text{“}String\text{”} \\ \text{“}Precio\text{”} & \text{“}Float\text{”} \\ \text{“}Duración\text{”} & \text{“}Integer\text{”} \end{cases}$$

3. La suscripción está formada por varias restricciones, y cada restricción está codificada de la siguiente manera:

$$\text{Restricción:} \begin{cases} Nombre & string \\ Tipo & \{\text{“}Integer\text{”}, \text{“}Float\text{”}, \text{“}String\text{”}\} \\ Operator & \begin{cases} Number & \{=, \neq, <, >, \geq, \leq\} \\ String & \{=, \neq, <, >, \geq, \leq, suffix, prefix, contains\} \end{cases} \\ Valor & \begin{cases} Integer & \{... \quad -3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad ...\} \\ Float & \{... \quad -0{,}2 \quad -0{,}11 \quad 0 \quad 1{,}0 \quad 3{,}0 \quad ...\} \\ String & \{\text{“}españa\text{”} \quad, \text{“}china\text{”} \quad, \text{“}italia\text{”} \quad, \text{“}pepe\text{”} \quad ...\} \end{cases} \end{cases}$$

A continuación se muestra una suscripción de un billete de avión, que requiere destino a España, origen en las ciudades de China, precio menor 500 euros y la duración menor 18 horas:

$$\text{Billete suscripción :} \begin{array}{l} \begin{array}{llll} \text{Nombre} & \text{Tipo} & \text{Operador} & \text{Valor} \end{array} \\ \begin{cases} \text{“}Destino\text{”} & \text{“}String\text{”} & \text{“}=\text{”} & \text{“}España\text{”} \\ \text{“}Origen\text{”} & \text{“}String\text{”} & \text{“}contains\text{”} & \text{“}China\text{”} \\ \text{“}Precio\text{”} & \text{“}Float\text{”} & \text{“}\leq\text{”} & \text{“}500\text{”} \\ \text{“}Duración\text{”} & \text{“}Integer\text{”} & \text{“}<\text{”} & \text{“}18\text{”} \end{cases} \end{array}$$

2.4. Términos del algoritmo de enrutamiento

Este apartado y los siguientes van a definir los objetos que están relacionados con el algoritmo *de enrutamiento,* y algunas relaciones sobre ellos que hemos definido como la cobertura . El objetivo es encontrar una manera para disminuir el tamaño del conjunto de las suscripciones en la tabla de *enrutamiento,* y creemos que eso mejora la eficiencia del algoritmo de conteo. Y al final vemos que la cobertura se sirve para eso.

2.4.1. Predicado

Los *predicados*[18] se utilizan para presentar si un atributo de la notificación satisface una restricción. Y se define como:

$$p(x) \begin{cases} true & x \in \{x \mid x\, op\, valor\} \\ false & x \notin \{x \mid x\, op\, valor\} \end{cases}$$

donde $op \in \{=, \neq, <, >, \geq, \leq, suffix, prefix, contains\}$ y $valor \in \{number\} \cup \{string\}$.

Ahora si un atributo cumple una restricción es equivalente a $p(a) = true$. Un ejemplo de la relación que nos interesa:

$\{precio\ de\ billete\ avión \mid precio > 100\} \supsetneq$
$\{precio\ de\ billete\ avión \mid precio > 200\} \supsetneq$
$\{precio\ de\ billete\ avión \mid precio = 200\}$

2.4.2. Filtro

El *filtro*[19] se usa para representar si una notificación satisface una suscripción. Y se define como: $f(x, y, z...) = p_1(x) \wedge p_2(y) \wedge p_3(z) \wedge ....$ Ahora si una notificación se cumple una suscripción es equivalente a $\exists\{a_1, a_2, ...\} \subseteq n,\ f(a_1, a_2, ...) = true$. Un ejemplo de la relación que nos interesa: $f_1 \rightarrow f_2 \rightarrow ..... \rightarrow f_{10} \rightarrow f_{11}$.

2.4.3. Cobertura y retículo

Ahora introducimos la relación importante que se ha descubierto entre los filtros, definiendo el concepto de la *cobertura*[20]. Después mostraremos que la cobertura es un *retículo*[21] en *álgebra abstracto*[22]. Por eso, es necesario conocer el retículo y demostrar que la cobertura es un retículo. Hay dos formas para deducir que la cobertura es un retículo especial. Uno es que la cobertura es un tipo de álgebra booleano y álgebra booleano es un retículo. Otro es demostrar que la cobertura se cumple todas las condiciones de un retículo. Y como en el sistema práctico, solamente necesitamos las características del *conjunto parcialmente ordenado*[23] de la cobertura, por eso aquí solamente demostramos que la cobertura es un *conjunto parcialmente ordenado*, y con eso ya es suficiente para apoyar nuestro trabajo.

2.4.3.1. Conjunto parcialmente ordenado

El conjunto parcialmente ordenado es *un sistema algebraico*[24] que está formado por un conjunto X con una relación binaria $\geq$ y cumple siguientes axiomas(para cualesquiera a, b, y c en X):

A_0: $a \geq a$.

A_1: $Si\ a \geq b\ y\ b \geq a\ , entonces\ a = b$.

A_2: $Si\ a \geq b\ y\ b \geq c\ , entonces\ a \geq b$.

Hay que dar cuenta para cualesquier a,b en X, se puede existir la relación, notamos $a \geq b$, o bién no existe la relación, notamos como $a \ngeq b$. También puede ser $a \geq b\ ,\ a \neq b$, notamos como

$a > b$.

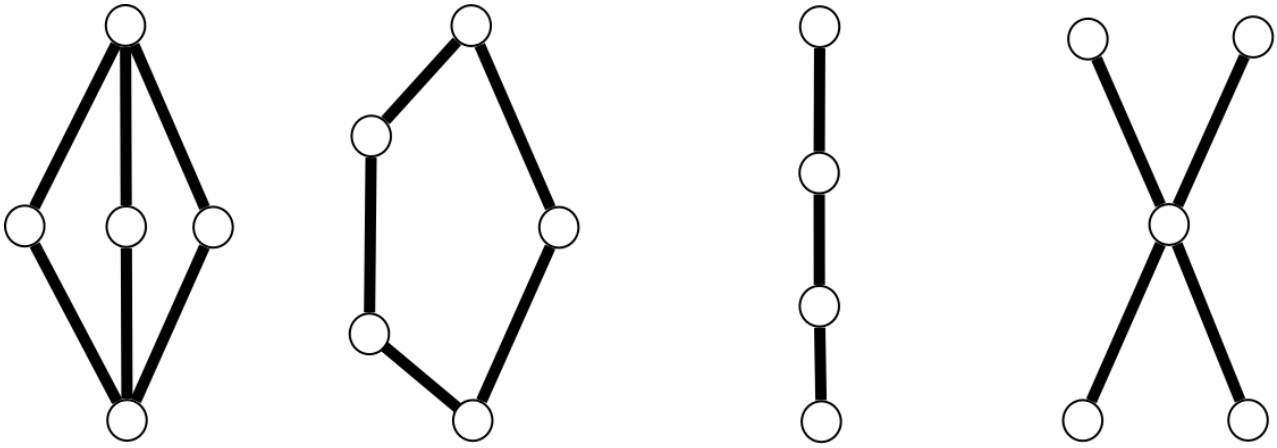

Figura 2.3: El ejemplo de los gráficos de conjunto parcialmente ordenado

En un conjunto X limitado, si $a > b$ y no existe u que $a > u > b$, llamamos a es una *cubierta mínima*[25] de b. Ahora para cualesquiera a,b en X, podemos conseguir una cadena como $a = u_1 > u_2 .... > u_i > u_{i+1} = b$. Con eso es suficiente decir que cualquier conjunto parcialmente ordenado se puede representar con un gráfico, también es decir que se puede implementar como una estructura de datos. Y para verificar la exactitud de la estructura, hay que probar que las cadenas de la relación se construyen correctamente. La implementación de la estructura es buscar algoritmos para mantener las cadenas.

2.4.3.2. Retículo

Sea S es un conjunto parcialmente ordenado, y $A \subseteq S$, para cada $a \in A$ y un $u \in S$, se cumple $u \geq a$, llamamos u es un *límite superior*[26] del conjunto A. Si v es *límite superior de* A, y para todos u *que son limite superior* de A, cumple $u \geq v$, llamamos v es *límite superior mínimo*[27] *de* A, notamos como *l.u.b(A)*[27]. Y similar si se cumple $a \geq u$, llamamos u es *límite inferior*[28] del conjunto A. v es *límite inferior máximo*[29] *de* A, notamos como *g.l.b(A)*[29].

Si L es un conjunto parcialmente ordenado, para cualquier de los dos elementos de L, si existe un *límite superior mínimo* y *límite inferior máximo* de ellos, llamamos L es un(a) (estructura) retículo.

2.4.3.3. Cobertura

Ahora vamos a definir la cobertura entre los filtros. Como he mencionado anteriormente, hay dos maneras para definir la cobertura, uno es definirla directamente a nivel de filtro, es decir si

$f_1 \rightarrow f_2$ decimos que f_2 cubre f_1, notamos como: $f_2 \geq f_1$. Y la estructura de esta relación forma álgebra booleana, y también se llama retícula booleana. La segunda forma es deducir la cobertura desde la cobertura entre los operadores, y ésto es lo que vamos a discutir. Hay dos razones para eso, uno es que fundamentalmente la cobertura de los filtros depende de los operadores, otro es que la relación entre los filtros y operadores es el punto de partida del nuevo algoritmo.

- **Cobertura de predicado**

  Si como predicado es $p(x)\begin{cases} true & x \in \{x \mid x\, op\, valor\} \\ false & x \notin \{x \mid x\, op\, valor\} \end{cases}$, el conjunto $X = \{x \mid p(x) = true\}$, decimos si p_1 cubre p_2 si y solo si $X_1 \supseteq X_2$, notamos $p_1 \geq p_2$. Según álgebra de conjunto, el $X_1 \supseteq X_2$, es un retículo, por eso $p_1 \geq p_2$ es un retículo. Para obtener más información, consulte las operaciones de conjuntos.

- **Cobertura de filtros**

  Sea $f_1(x, y, z...) = p_1(x) \wedge p_2(y) \wedge p_3(z) \wedge ...$ y $f_2(x, y, z...) = q_1(x) \wedge q_2(y) \wedge q_3(z) \wedge ...$, $f_1 \geq f_2$, si y sólo $\forall p_{f_1}$, $\exists q_{f_2}$ que $p_{f_1} \geq q_{f_2}$, así que cumple:

  - **A_0**: $f_1 \geq f_1$, porque $\forall p_{f_1}$, $\exists p_{f_1}$ que $p_{f_1} \geq p_{f_1}$.

  - **A_1**: $si\ f_1 \geq f_2\ y\ f_2 \geq f_1\ ,\ entonces\ f_1 = f_2$, porque $\forall p_{f_1}$, $\exists q_{f_2}$ que $p_{f_1} \geq q_{f_2}$, y $\forall p_{f_2}$, $\exists q_{f_1}$ que $p_{f_2} \geq q_{f_1}$, por eso $\forall p_{f_1}$, $p_{f_1} \geq q_{f_2} \geq p_{f1}$, y $p_{f_1} = q_{f1}$(nota: que cada predicado de un filtro representa una restricción distinta, por eso entre ellos no hay cobertura. Es decir no existe $p_1 \geq p_2$ y $p_1 \neq p_2$ en un filtro).

  - **A_2**: $si\ f_1 \geq f_2\ y\ f_2 \geq f_3\ ,\ entonces\ f_1 \geq f_3$, porque $\forall p_{f_1}$, $\exists q_{f_2}$que $p_{f_1} \geq q_{f_2}$, y $\forall q_{f_2}$, $\exists r_{f_3}$ que $q_{f_2} \geq r_{f_3}$, por eso $\forall p_{f_1}$,$\exists r_{f_3}$ que $p_{f_1} \geq r_{f_3}$.

  Por lo tanto, la cobertura de filtros es un conjunto parcialmente ordenado.

2.4.4. Algoritmo de enrutamiento

Ya hemos visto los términos para el *algoritmo de enrutamiento*, y ahora vamos a explicar su objetivo y procesamiento que está aplicado sobre esos términos. Como hemos visto en la figura 2.2, el núcleo del mensaje filtrado es seleccionar las interfaces para reenviar el mensaje, y eso es nuestro objetivo y se define como:

$$Seleccción(n) = \{\, i \mid i \in I \wedge \exists < i, f > \ f(n) \rightarrow true\}$$

$$\begin{cases} n & notificación \\ i & interface \\ I & todos\, interfaces\, del\, este\, broker \\ f & filtros \\ < i, f > & el\, filtro\, f\, que\, viene\, de\, la\, interface\, i \end{cases}$$

El objetivo del proceso de selección es encontrar todas las interfaces que hay que reenviar la notificación tan pronto como sea posible, Como se veía en la definición, si la notificación satisface algunas suscripciones que vienen de una interfaz, entonces hay que enviar la notificación a esta interfaz. Por eso, es necesario crear una estructura, y ofrecer el almacenamiento a las suscripciones y a la interfaz que se ha asociado, en nuestro caso, está utilizando la tabla de *enrutamiento.*

1. **Tabla de *enrutamiento***

   Como se ve en la figura 2.4, en la tabla de *enrutamiento* se guardan los predicados (restricciones), filtros (suscripciones), interfaces y las relaciones entre ellos. Ahora la selección de las interfaces de reenvío del mensaje se simplifica como:

$$Seleccción(n) = \{\, i \mid i \in I \wedge \exists < i, f > que\, \forall p_f \rightarrow True \,\}$$

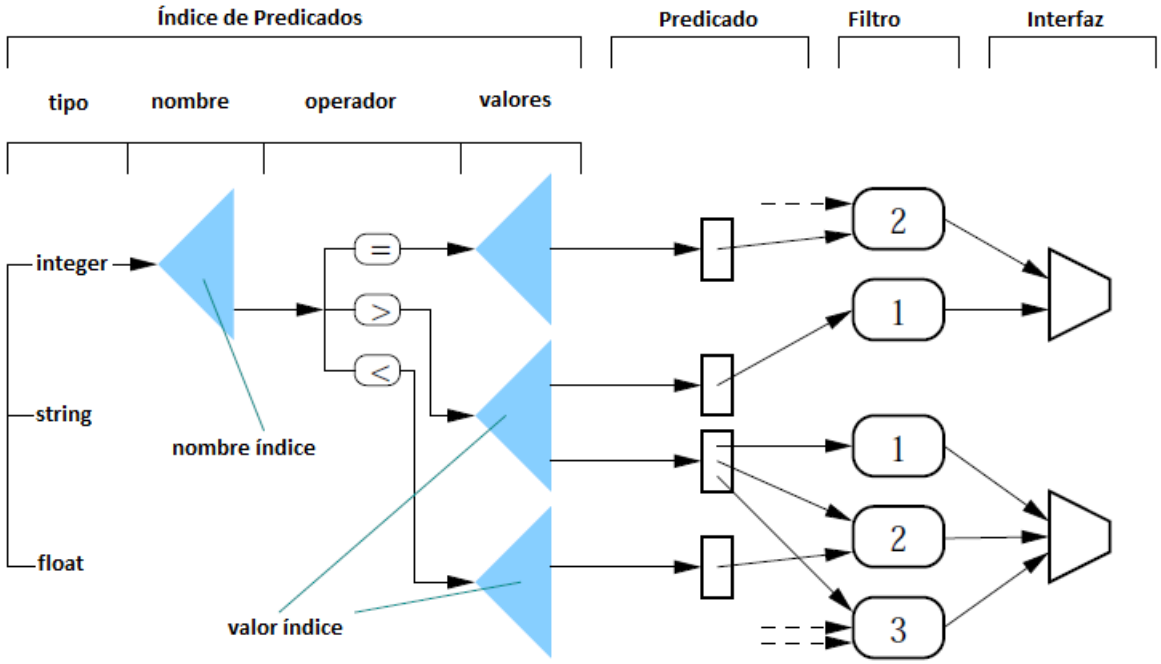

Figura 2.4: La estructura de la tabla de enrutamiento

A continuación, mostramos el algoritmo implementado sobre la tabla de *enrutamiento.*

1. **algoritmo de conteo**

   La figura 2.1 muestra el pseudo-código detallado del algoritmo de conteo, y su proceso está definido como:

   - **Paso 1.** Seleccionamos todos predicados que cumplen con algún atributo de la notificación, $Seleccción(p) = \{\, p \mid \exists a \, que \, p(a) \rightarrow True\}$

   - **Paso 2.** Seleccionamos todos los filtros que están relacionados con el predicado, y contar el número que aparece, es decir, $Seleccción(< f, c >) = \{< f, c > |\ c = número \, de \, p \, , \, donde \, p \, está \, selecionado \ y \ p \in f \}$

   - **Paso 3.** Seleccionamos todas interfaces de los filtros que están en el conjunto de resultado, con la condición que el número contado de sus predicados es **igual** que su número total de predicados. Esa condición aplica que todos los predicados del filtro indicado han aparecido y están satisfechos con la notificación, y indica que el filtro está satisfecho con la notificación.

2.4.5. Algoritmo de enrutamiento y la cobertura

En teoría, el algoritmo de *enrutamiento* (*el algoritmo de conteo*) hay que seleccionar los filtros que cumplen con la notificación, y enviar el mensaje a las interfaces que indican esos filtros. Pero por la misma interfaz, sólo hace falta enviar una vez el mensaje. Eso es decir para cada interfaz, si sabe que hay un filtro que cumple con la notificación, ya podemos reenviar el mensaje a esta interfaz. Y en el otro lado, sabemos si el fitro1 cubre el filtro2, púes los mensajes que pasan filtro2, seguro pasan filtro1. Así si filtro1 y filtro2 están asociados en la misma interfaz, sólo es necesario verificar si el mensaje pasa filtro1, no hace falta verificar los dos. Por esta razón, si por cada interfaz, usamos un conjunto mínimo de filtros, que cubren los restos filtros, podemos reducir el tamaño total del filtro(tamaño total es la suma de los conjuntos por cada interfaz).

Al final, voy a insistir otra vez, si usamos el conjunto mínimo de filtros para reducir el tamaño de los filtros para el algoritmo de *enrutamiento,* el coste para calcular el conjunto mínimo disminuye la ganancia que hemos conseguido. Y por eso hay que evaluar el coste y la ganancia, para saber si realmente puede mejorar el sistema.

Algoritmo 2.1 algoritmo de conteo

proc selección(notificación n)

```
map <filtro,int> contadores = {Ø} ;
set<interfaz> resultado = {Ø};
set<predicado> predicados = {Ø}

//Paso 1:

foreach a in n
    predicados = predicados ∪{ p | p(a) → true }
end foreach

//Paso 2:

foreach p in predicados
    foreach f in p.filtros

        if f ∉contadores
            contadores = contadores ∪ { < f, 0 > }
        end if

        contadores[f ]= contadores[f]+1;

    end foreach
end foreach

//Paso 3:

foreach <f,c>∈ contadores

    if c == f .size
        resultado= resultado ∪ f.interfaces
    end if

end foreach

return result
```

end ***proc***

Capítulo 3

Desarrollo de la cobertura

En este capítulo, se describe el desarrollo de la cobertura. En el primer apartado vamos a explicar la tabla de cobertura general y sus operaciones. Enseguida, damos la tabla mejorado, cual usa el algoritmo de conteo para calcular la cobertura, y sus estructuras necesarias. Al final damos una visión sobre la implementación completa.

Como ya sabíamos que la cobertura de filtros forman *un conjunto parcialmente ordenado*, y por eso se puede representar como un grafo, la figura 3.1 muestra un ejemplo. Así podemos implementar una tabla a nivel software que representa la cobertura y llamamos la tabla de cobertura.

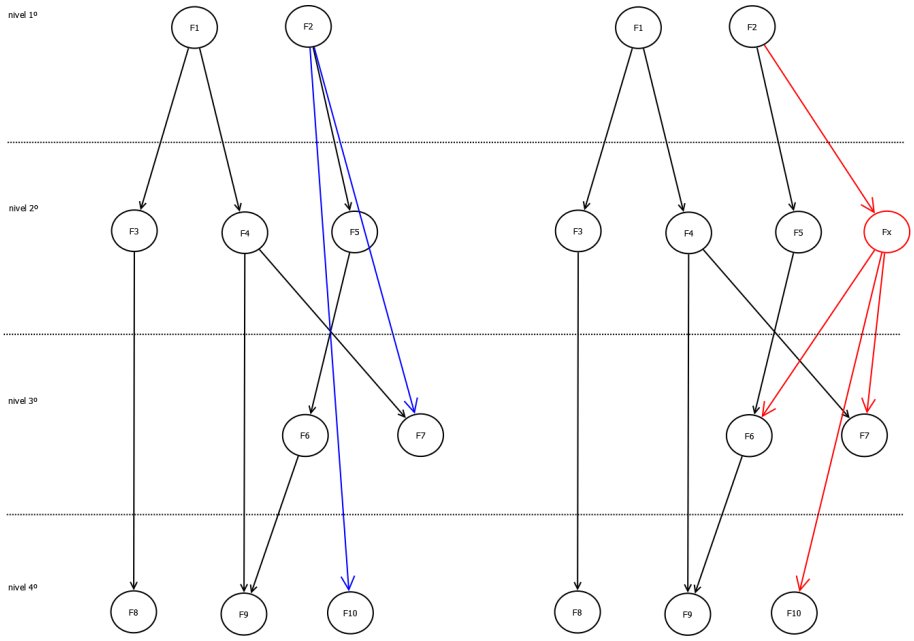

Figura 3.1: El ejemplo de la estructura de la cobertura de los filtros

3.1. Tabla de la cobertura para filtros

Una tabla de cobertura es una estructura de datos para guardar la cobertura de los filtros añadidos, tiene que proporcionar la operación de añadir un filtro a la tabla y la operación de quitar un filtro desde la tabla. En ambos casos, tenemos que modificar la cobertura entre los filtros. También, podemos decir que una tabla de cobertura es una estructura a nivel programación que equivale a *un conjunto parcialmente ordenado.* Para implementar una estructura así, la dificultad es como calcular la cobertura entre los filtros en cuando añade y quita un filtro. La figura 3.1 muestra la cobertura de la tabla cuando esté el filtro Fx y cuando no esté él. Pues si vamos a añadir el filtro Fx, el primer paso es quitar los enlaces que son azules, después añadimos los enlaces rojos y al contrario en quitarlo. Así que, el problema consiste en cómo encontrar los filtros que tienen conexión con el filtro indicado y como se modifican los enlaces entre estos filtros según la operación. En general, se usa la teoría *cubierta mínima,* cual está mencionado en el capítulo 2, para encontrar los filtros que tienen conexiones directa al filtro indicado.

- **Operación de añadir un filtro a la tabla**

  La operación de añadir un filtro a la tabla, ocupa añadir el nuevo filtro a la tabla y renovar los enlaces entre los filtros. Se define su proceso como:

Algoritmo 3.1 Algoritmo Add

```
proc add(filtro f)
    foreach filter in table
        if (filter es cubierta mínima de f)
            Link(filter,f)
        endif
        if (f es cubierta mínima de filter)
            Link(f,filter)
        endif
    end foreach
    foreach filterUpper in upperLinked(f)
        foreach filterLower in lowerLinked(f)
            if (filterUpper linked with filterLower)
                DesLink(filterUpper,filterLower)
            endif
    end foreach
end proc
```

- **Operación de quitar un filtro desde la tabla**

  Similar el añadir, el algoritmo de operación de quitar puede estar definido como:

Algoritmo 3.2 Algoritmo Remove

```
proc remove(filtro f)
    foreach filterUpper in upperLinked(f)
        foreach filterLower in lowerLinked(f)
            if (filterUpper NotOtherLinked with filterLower)
                Link(filterUpper,filterLower)
            endif
    end foreach
    foreach filterUpper in upperLinked(f)
        DesLink(filterUpper,f)
    end foreach
    foreach filterLower in lowerLinked(f)
        DesLink(f,filterLower)
    end foreach
end proc
```

3.2. Tabla de la cobertura usando el algoritmo de conteo

En el proceso de añadir, para encontrar los filtros que son las *cubiertas mínimas* del filtro que va a añadir, y los filtros que sus *cubiertas mínimas* son el filtro que va a añadir (es decir que tenemos que encontrar todos los conexiones directas al filtro que va a añadir), hay que recorrer todos los filtros en la tabla y esto produce un coste muy grande de tiempo. Para salvar este caso, usamos la cobertura de predicados y el algoritmo de conteo para encontrar los filtros que están relacionados (cubren o están cubiertos) con el filtro que va a añadir, después minimizamos los filtros encontrados (eliminar los filtros que no tienen relación directa al filtro que va a añadir) y obtenemos los filtros que existen enlaces directos al filtro que va a añadir. La minimización se parece el proceso general de añadir, pero ha sustituido el conjunto total de filtros con lo que ha producido el algoritmo de conteo. En el otro lado, como ya sabemos $f_1 \geq f_2$, si y sólo si $\forall p_{f_1}$, $\exists q_{f_2}$ que $p_{f_1} \geq q_{f_2}$ así que f_1 tiene menos predicados que f_2, por lo tanto, definimos $SelecciónInferior(f_x) = \{ f \mid \forall p_{f_x}, \exists q_f \; que \; p_{f_x} \geq q_f \}$ para encontrar todos los filtros que están cubiertos del filtro f_x y $SelecciónSuperior(f_x) = \{ f \mid \forall p_{f_x}, \exists q_{f_x} \; que \; p_f \geq q_{f_x} \}$ para encontrar todos los filtros que cubren el filtro f_x. Con las dos selecciones, renovamos el proceso del algoritmo de conteo.

3.2.1. Algoritmo de conteo para la cobertura de filtros

1. **Algoritmo de conteo para la cobertura**

   La figura 3.3 muestra el pseudo-código detallado del algoritmo de conteo, y se define como:

   - **Paso 1.** Seleccionamos todos los predicados que cubren algunos predicados del filtro que vamos a añadir y los guardamos. También seleccionamos todos predicados que está cubiertos y los guardamos.

Algoritmo 3.3 Algoritmo de conteo para la cobertura de filtros

```
proc selecciónCobertura(filtro f_X)
    map <filtroInf ,int> contadorInf = {Ø} ;
    map <filtroSup ,int> contadorSup = {Ø} ;
    set<filtro> resultadoInf = {Ø};
    set<filtro> resultadoSup = {Ø};
    set<predicado> predicadoInf = {Ø}
    set<predicado> predicadoSup= {Ø}

//Paso 1:

    foreach p_fx in f.predicados
        predicadoInf = predicadoInf ∪{ p | p_fx ≥ p }
        predicadoSup = predicadoSup ∪{ p | p ≥ p_fx }
    end foreach

//Paso 2:

    foreach p in predicadoInf
        foreach f in p.filtros
            if f ∉contadorInf
                contadorInf = contadorInf ∪{<f,0>}
            end if
            contadorInf[f]= contadorInf[f]+1;
        end foreach
    end foreach

    foreach p in predicadoSup
        foreach f in p.filtros
            if f ∉contadorSup
                contadorSup = contadorSup ∪ { <f,0>}
            end if
            contadorSup[f]= contadorSup[f]+1;
        end foreach
    end foreach

//Paso 3:

    foreach <f,c>∈ contadorSup
        if c =f.size and f.size ≤ f_X.size
            resultadoSup= resultadoSup ∪ f
        end if
    end foreach

    foreach <f,c>∈ contadores
        if c= f_X.size and f.size ≥ f_X.size
            resultadoInf= resultadoInf ∪ f
        end if
    end foreach

    return <resultadoInf,resultadoSup>

end proc
```

- **Paso 2.** Seleccionamos todos los filtros que están relacionados con los predicados, y contar el número que aparecen, es decir, $Seleccción(< f, c >) = \{< f, c > |\ c = número\, de\, p_f\, aparecido\}$

- **Paso 3.** Seleccionamos los filtros que están en el conjunto superior con la condición que el número de su predicado contado es **igual** que su número de predicados, además su número de predicados es **igual o menor** que el número de predicados del filtro que va a añadir. Con eso, aseguramos que todos los filtros seleccionados cubren el filtro indicado por la razón de que todos sus predicados cubren algún predicado del filtro indicado. En el otro caso, seleccionamos los filtros inferiores con la condición que su número de predicados es **igual o mayor** que lo del filtro que va a añadir y el número contado es **igual** que lo del filtro que va a añadir.

3.3. Desarrollo de la tabla de conteo

La tabla de conteo está formado por sus interfaces de los operaciones y su estructura de datos. La figura 3.2 muestra su implementación que es la clase "LatticeCountingTableImpl". Sus interfaces que viene de la tabla general de cobertura ("LatticeTable") están definidos como:

- **add:** añadir un filtro a la tabla, es el lugar donde vamos a aplicar el algoritmo de conteo para los filtros, y usamos la función "buildCoverage" para renovar la cobertura cuando añadimos un nuevo filtro a la tabla.

Algoritmo 3.4 Add with counting

proc add(filtro f)

<resultadoInf,resultadoSup>=selecciónCobertura(filro f)
<resultadoMaxInf,resultadoMinSup>=minimize(<resultadoInf,resultadoSup>)
construirCobertura(filtro f, resultadoMaxInf,resultadoMinSup)

end ***proc***

- **remove:** quitar un filtro desde la tabla.

- **clear:** eliminar todos elementos desde la tabla.

- **lowers:** devuelve el conjunto de filtros que están cubiertos por el filtro indicado. Sólo para los filtros que están en la tabla, si no genera una excepción.

- **uppers:** devuelve el conjunto de filtros que cubren el filtro indicado. Sólo para los filtros que están en la tabla, si no genera una excepción.

- **greatLowers:** devuelve el conjunto de filtros que el filtro indicado es su *cubierta mínima*. Sólo para los filtros que están en la tabla, si no genera una excepción.

- **leastUppers:** devuelve el conjunto de filtros que son *cubiertas mínimas* del filtro indicado. Sólo para los filtros que están en la tabla, si no genera una excepción.

Sus atributos están definidos como:

- **conn:** indica la conexión a la que está asociado.
- **externalIndex:** donde guarda el conjunto mínino de filtros.
- **internalIndex:** donde guarda el conjunto total de filtros.
- **lattice:** donde se guardan los nodos de cobertura.

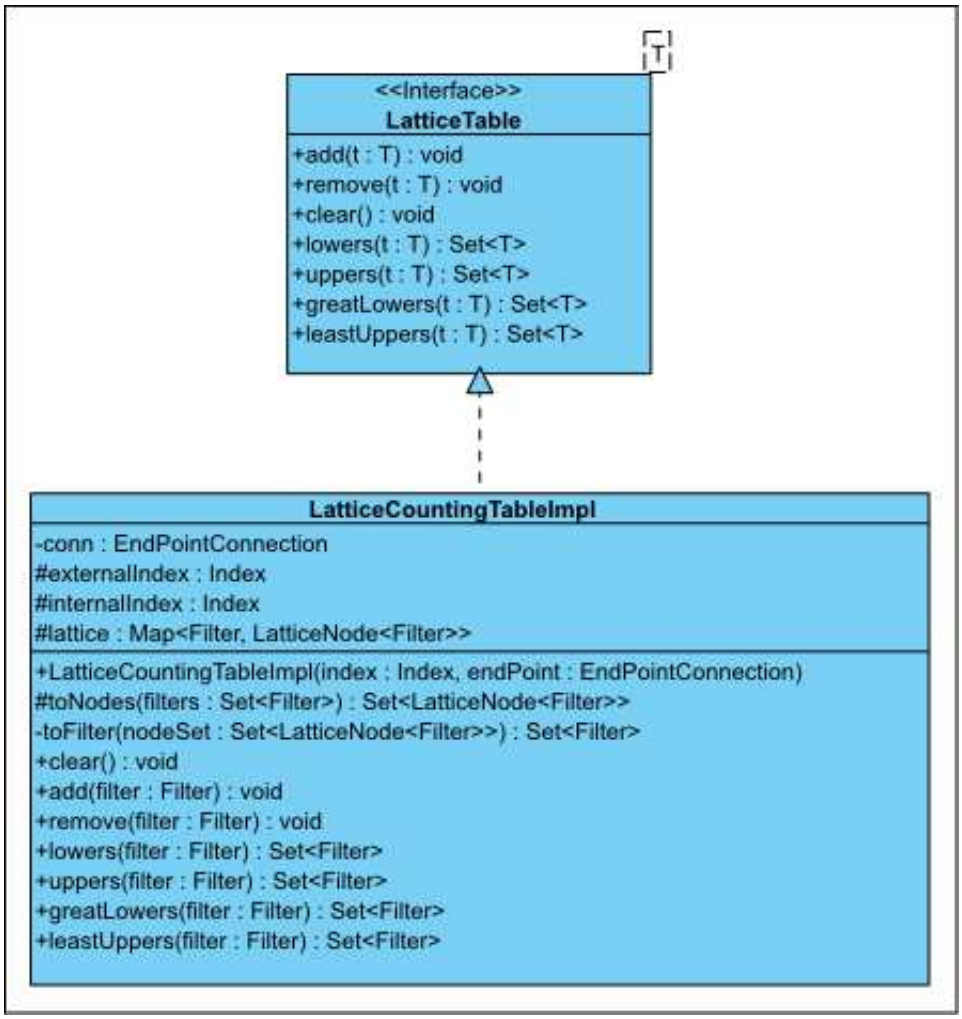

Figura 3.2: El diagrama de clase de la tabla de conteo

3.3.1. Desarrollo de la estructura interna

Los nodos son donde se guardan y mantienen los enlaces de la cobertura, la figura 3.3 muestra su implementación. En general, la clase “LatticeNode” ofrece las interfaces para modifcar los enlaces entre los nodos, y guarda los nodos que tienen una relación con este nodo en directo, como en indirecto.

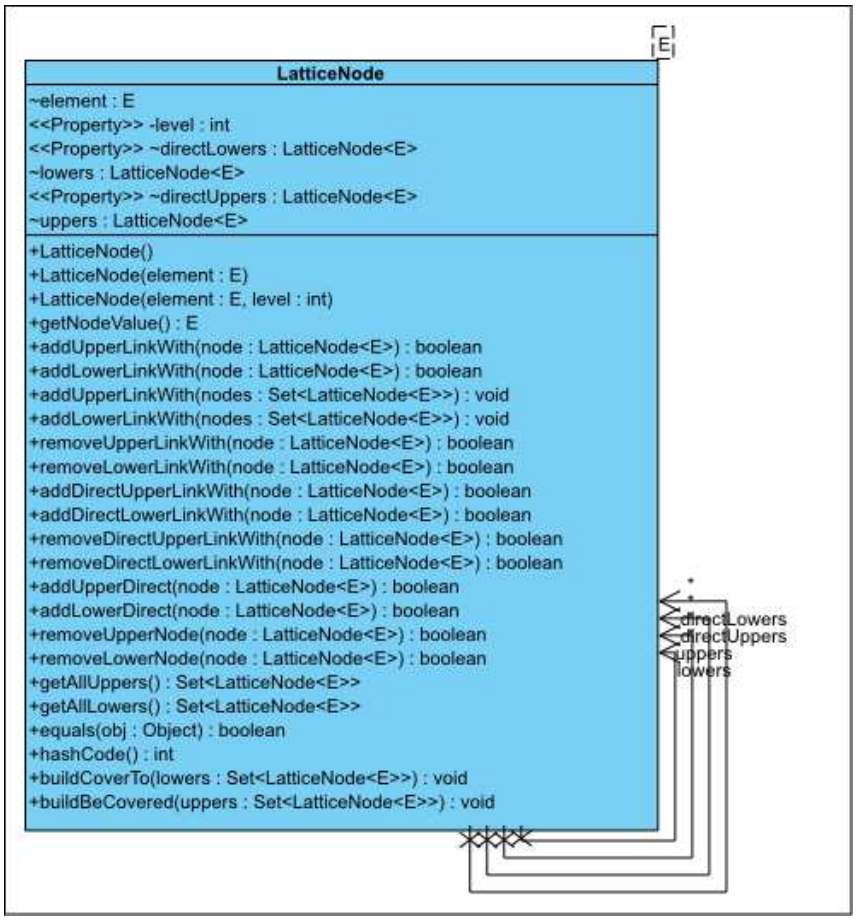

Figura 3.3: El diagrama de clase del nodo de la tabla de conteo

- **Las operaciones de modificar los enlaces:** esas operaciones sólo modifican un enlace desde este nodo con el nodo indicado.
- **Las operaciones de construir relación de la cobertura:** esas operaciones modifican los enlaces desde este nodo con el nodo indicado, y si es necesario támbien modifican los enlaces entre otros nodos para matener la cobertura correcta entre los nodos.

3.3.2. Desarrollo del proceso de la construcción de la cobertura

Las operaciones para el proceso de añadir filtro está muestrado por la figura 3.4

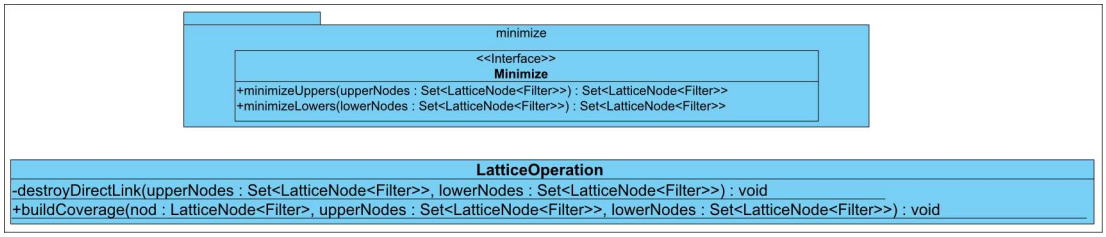

Figura 3.4: El diagrama de clase de la construcción de la cobertura

3.3.3. Desarrollo de la estructura de soporte

La “CountingIndex”, que ha heredado de la clase ”AbstractIndex´´ con la interfaz “Index”, es la estructura donde relaciona predicados y filtros. El nuevo algoritmo de conteo se implementa como un método “matchedCoverageFilters” en esta clase, la figura 3.5 muestra su implementación.

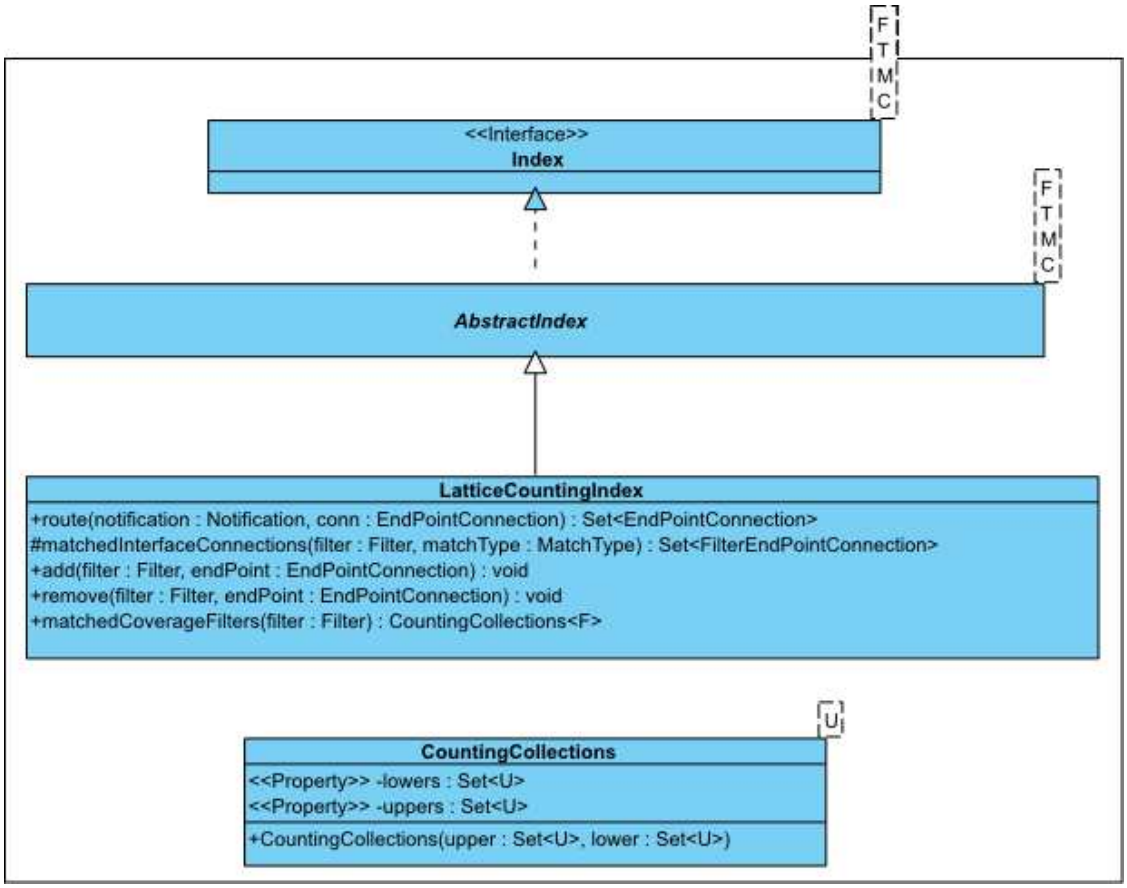

Figura 3.5: El diagrama de clase del índice de conteo

3.3.4. Desarrollo de la cobertura de los predicados

Para poder soportar el nuevo algoritmo hay que desarrollar la cobertura de los predicados y está descrito en la figura 3.6.

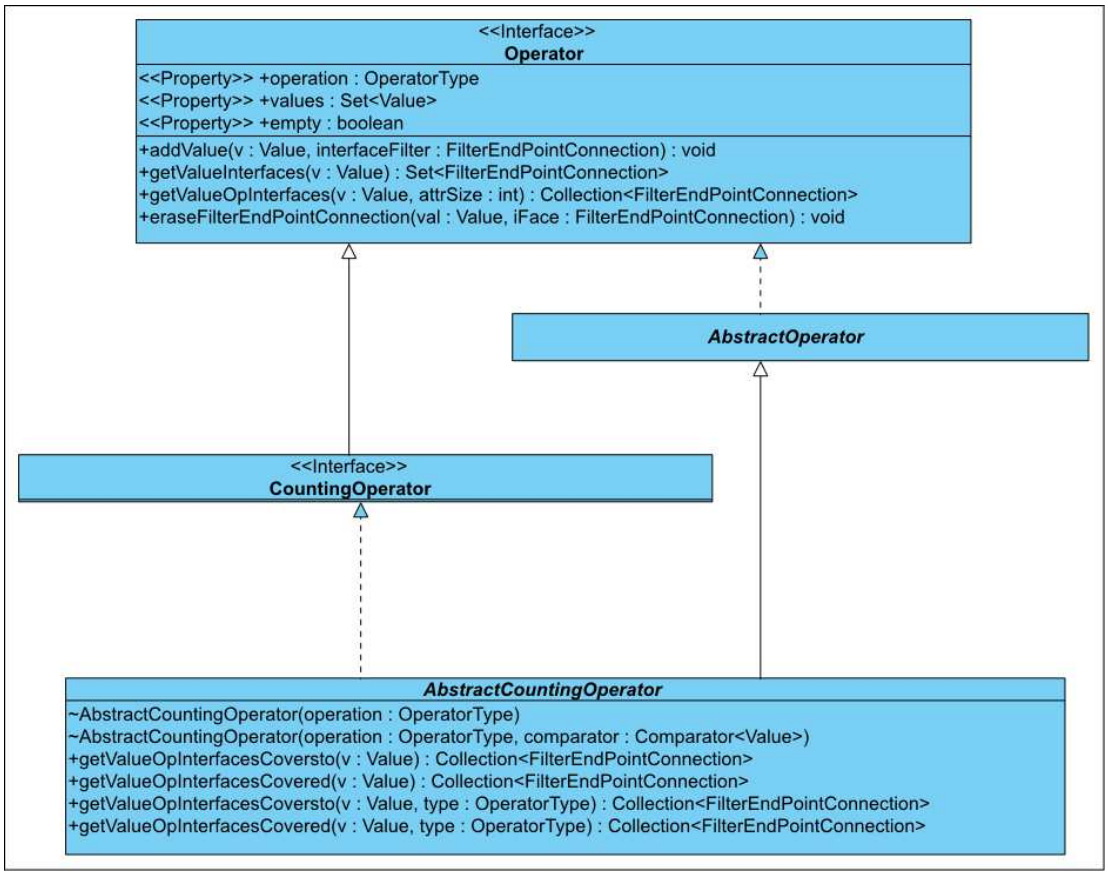

Figura 3.6: El diagrama de clase de los operadores del predicado

3.3.5. Diagrama de paquetes

Al final muestra un visión global de nuestro desarrollo y está en la figura 3.7

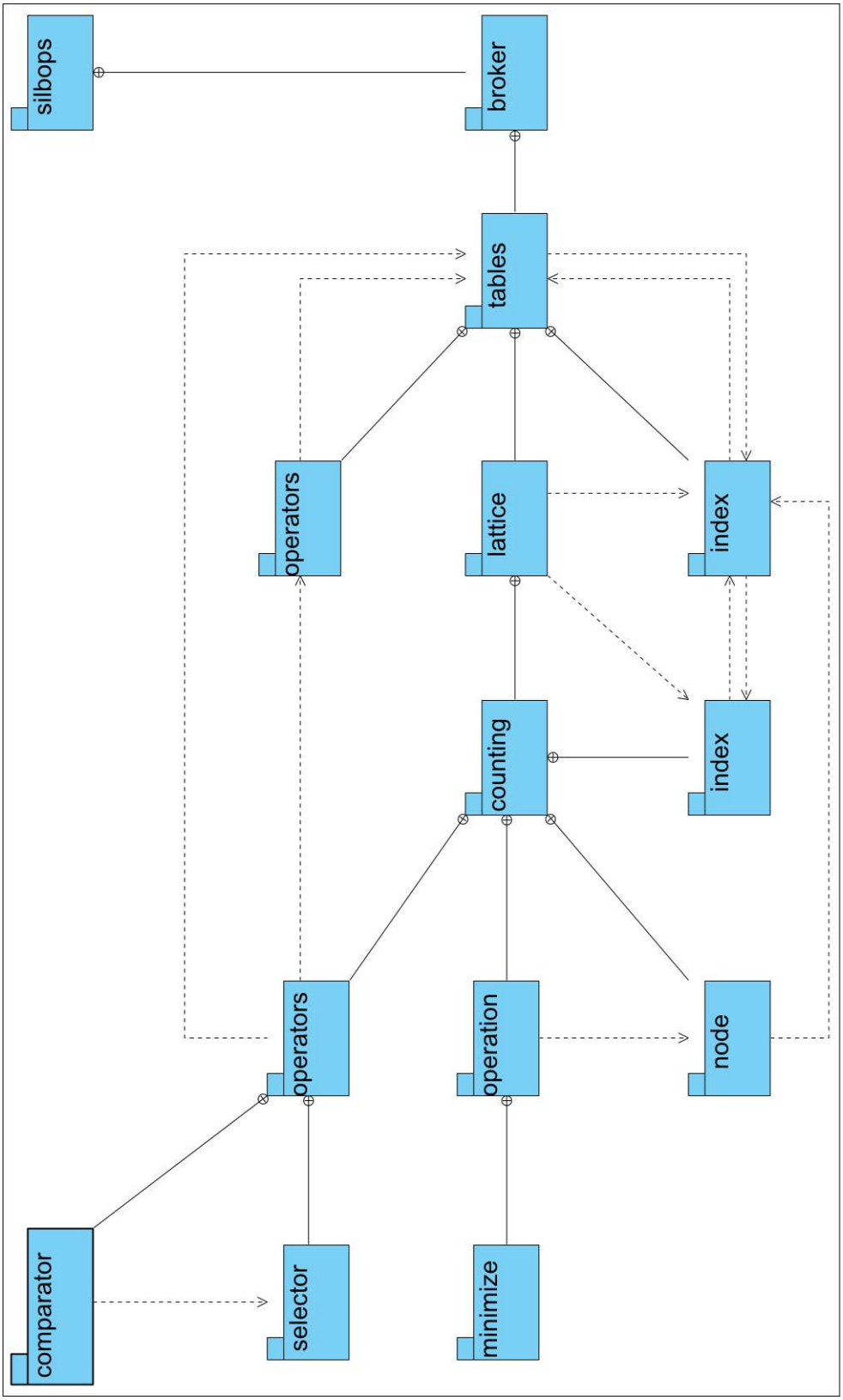

Figura 3.7: El diagrama de paquetes del proyecto

Capítulo 4

Evaluación del rendimiento de las dos tablas

Con el fin de comparar las dos tablas de una forma precisa, hemos usado el *test parametrizados en JUnit*[30] y pasamos la tabla y una lista de filtros como los parametros al sistema para medir el tiempo de cada operación. En nuestro caso, aseguramos que las dos tablas procesan las mismas listas de filtros, el tiempo está medido con función de Java basado en una unidad de nanosegundo y los resultados se muestran en microsegundo y con un entorno de núcleo de linux 3.2.0 y Intel Core 2 6300, 1.800 MHz, 2 GB de DDR2.

- **Descripción de la tabla general**

  La tabla general que vamos a comparar está implementado como la tabla de retículo y ha aplicado las optimizaciones según el artículo “Representing Large Concept Hierarchies Using Lattice Data Structure”.

- ***Test parametrizados en JUnit***

  Los tests parametrizados son una forma de escribir un test genérico y poder correrlo con juegos de datos diferentes. En nuestro caso, esos tests están creados con la herramienta *JUnit*.

4.1. Variables independientes y dependientes

El variable independiente es el número de filtro de la lista. En nuestro caso, se generan los filtros según una distribución *Zipf*[31] con un tamaño de sequencia de logaritmo de 10 pasos, y medimos el tiempo total de una operación sobre esta lista.

4.2. Resultado de operación añadir

La figura 4.1 muestra el tiempo que tarda la operación añadir con distintos tamaños de filtros. La linea verde describe el tiempo de la tabla general con la cobertura, y el rojo indica el tiempo de la tabla de conteo.

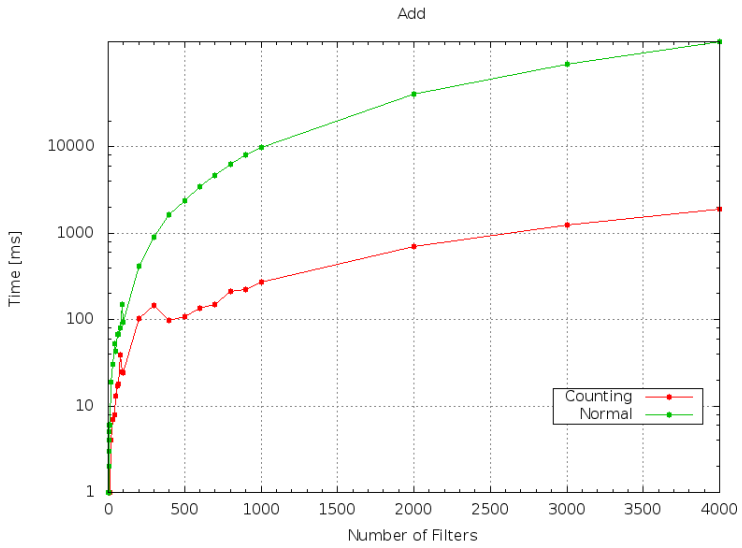

Figura 4.1: El tiempo de la operación añadir

La figura 4.2 mustra la zona confusa de la figura 4.1.

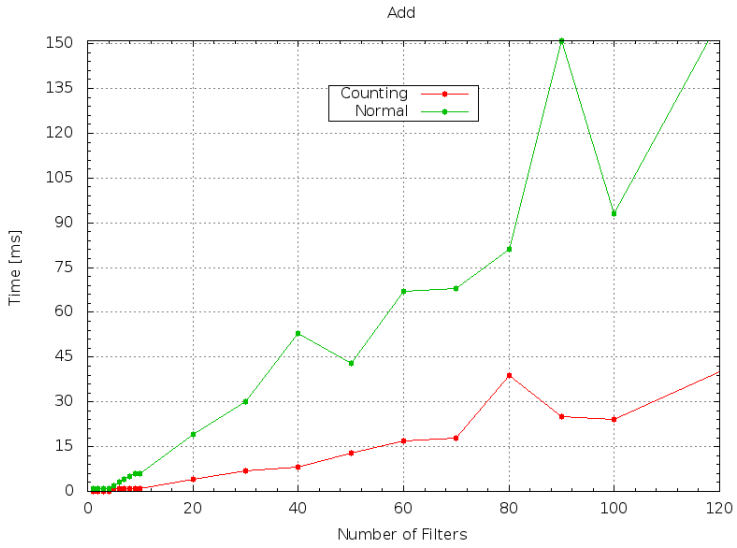

Figura 4.2: El tiempo de la operación añadir en la zona confusa (menor 120 filtros)

4.2.1. Conclusiones

Según el gráfico 4.1 y 4.2 sabemos que la tabla de conteo siempre es más rápido en añadir que la tabla general. Cuando añadimos más de mil filtros, la diferencia del tiempo ya ha llegado hasta los 9700 microsegundos y cuando llegan dos mil filtros, la ganancia ya es más de 20000 microsegundos.

4.3. Resultado de operación quitar

La figura 4.3 muestra el tiempo que tarda la operación quitar con distintos tamaños de filtros. La linea verde describe el tiempo de la tabla general con la cobertura, y el rojo indica el tiempo de la tabla de conteo.

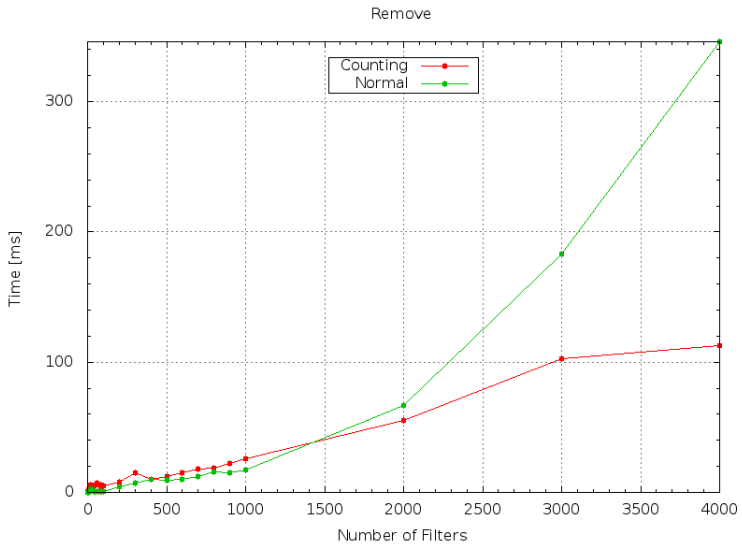

Figura 4.3: El tiempo de la operación quitar

La figura 4.4 mustra la zona confusa de la figura 4.3.

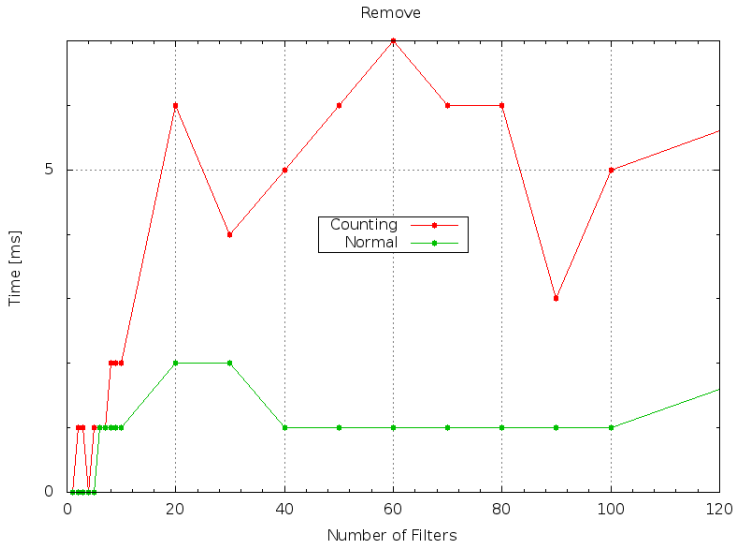

Figura 4.4: El tiempo de la operación quitar en la zona confusa (menor 120 filtros)

4.3.1. Conclusiones

En este caso, según el cuadro 4.3 y 4.4 , podemos ver que la diferencia entre los dos tablas no es tan grande como en el caso de añadir, y notamos antes de 1500, que la tabla general funciona más rápido, pero obtiene una ganancia menor que 10 micro segundo, después del 1500, la tabla de conteo siempre es más rápida.

4.4. Conclusión final

A pesar de que la tabla general es más rápida en la operación de quitar entre 0 a 1500, la tabla de conteo es más rápida en cualquier de estos casos. Hay que darse cuenta que el tiempo que gana la tabla general en quitar es muy pequeño y al contrario el tiempo que pierde en añadir es bastante grande. Así, si integramos todos los casos, podemos decir la tabla de conteo tiene más ventaja que la tabla general.

Capítulo 5

Evaluación del impacto sobre la mejor tabla frente al sistema general

En primer lugar, me gustaría destacar una vez más que el propósito de este artículo es evaluar el impacto sobre la tabla de cobertura frente al sistema práctico. El rendimiento del sistema práctico fundamentalmente depende el algoritmo de enrutamiento y el conjunto del destinatario del enrutamiento. En nuestro caso, los dos factores son el algoritmo de conteo y el conjunto de filtros. Por eso el coste de tiempo depende el tiempo de dirigir una notificación y la gestión de los filtros. Para evaluar el impacto, nuestra idea es medir los tiempos de la gestión de los filtros con el caso sin cobertura y el caso con cobertura, tanto como en el tiempo de enrutamiento de una notificación. Después, hallamos las diferencias de tiempo entre los dos casos en la gestión de los filtros, tanto como en enrutamiento de una notificación. Al final, calculamos el ratio entre el filtro y la notificación para saber con cuantos notificaciones el sistema con cobertura puede ganar el sistema sin cobertura en soló gestionar un filtro. Para conseguir las medidas, hemos usado la tabla de enrutmiento con el conjunto total de filtros y el conjunto comprimido de filtros con la cobertura para el tiempo de enrutmiento de la notificación y el tiempo de cada operación sobre los filtros.

5.1. Resultados básicos

La figura 5.1 muestra el tiempo que ha medido según distintos tamaños de filtros en el sistema con cobertura, tanto como en el sistema sin cobertura. Como vemos, la línea verde siempre supera la línea roja, por lo tanto, resultamos que el sistema con la cobertura siempre es más rápido en procesar las notificaciones que el sistema sin cobertura.

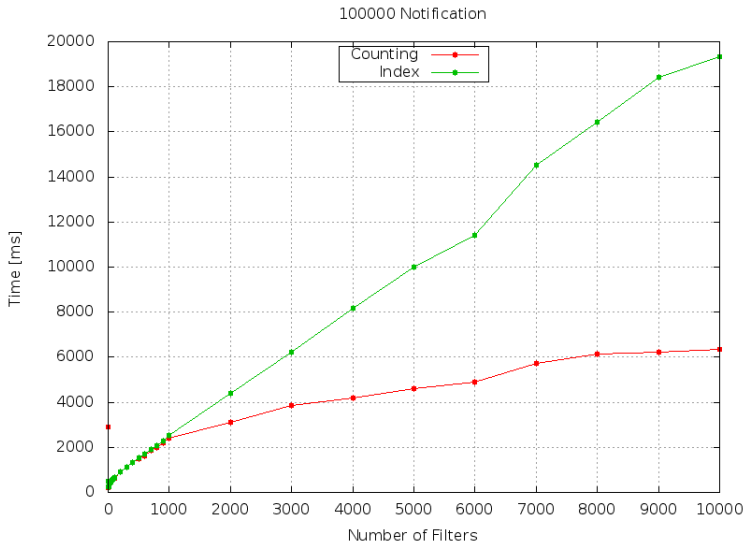

Figura 5.1: El tiempo de procesar 100000 notificaciones según número de filtros

La figura 5.2 y 5.3 muestra el coste de tiempo del sistema con la cobertura, tanto como en el sistema sin la cobertura. Notamos que cuando con más de mil filtros, la diferencia de tiempo ya es bastante grande en la operación de añadir, y crece rápido según crece el número de los filtros. En el otro lado, la diferencia del tiempo de la operación de quitar no es tan grande, y crece levemente. En el caso menor de mil, la situación es un poco complicado, sólo podemos decir que la diferencia de tiempo total de dos operaciones no va a superar los 1000 microsegundos.

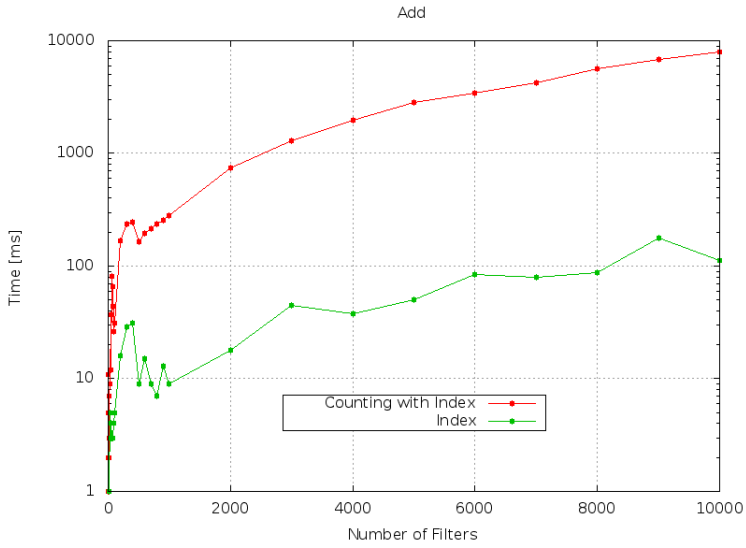

Figura 5.2: El tiempo de la operación de añadir un conjunto de filtros al sistema

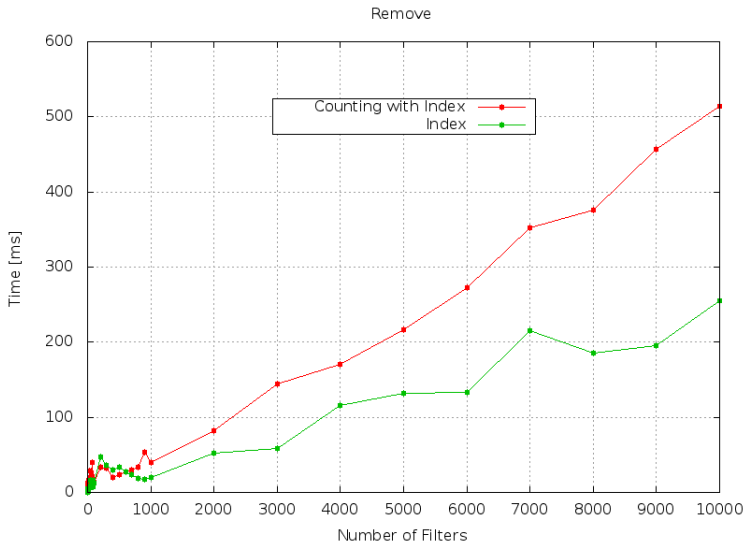

Figura 5.3: El tiempo de la operación de quitar un conjunto de filtros del sistema

5.2. Resultado del ratio entre el filtro y la notificación

Para obtener el resultado final, es muy importante medir bien el tiempo que tarda en gestionar un filtro según sistema con distintos tamaños de filtros, y el tiempo de procesar una notificación en esa base de filtros. Porque la cobertura varía según existen distintos filtros en el sistema, y eso afecta el tiempo de procesar una notificación (El tiempo que ha reducido con la cobertura varía). Así que hemos diseñado un nuevo experimento para obtener el ratio entre el filtro y la notificación. Los tiempos medidos son los tiempos de operaciones del filtro y el tiempo de procesar las notificaciones. Las tablas 5.1 y 5.2 muestran los parámetros de nuestro experimento. En general, hemos medido un promedio del tiempo para añadir y quitar con varios filtros, y tamaño de los filtros se cambia según los filtros que hay en el sistema. Por ejemplo, cuando el sistema obtiene 1000 filtros, añadimos 100 filtros al sistema, y luego quitamos esos 100 filtros. Y el tiempo promedio de gestionar un filtro en la base de 1050 filtros es la suma del tiempo de añadir eso 100 filtros y quitarlos dividido por 100. En el otro lugar. El tiempo de procesar una notificación siempre es una promedio del tiempo de procesar 100000 notificaciones. Y luego hallamos el tiempo diferencia del gestionar un filtro, tanto como en procesar una notificación, entre el sistema con cobertura y sin cobertura.

Rango de filtros en el sistema	tamaño de paso(N)	tiempo de operación del filtro($\frac{T_{total}}{N}$)
0 -200	10	T= $\frac{T_{10}}{10}$
200-1000	50	T= $\frac{T_{50}}{50}$
1000-4000	100	T= $\frac{T_{100}}{100}$
4000-10000	500	T= $\frac{T_{500}}{500}$

Cuadro 5.1: Tabla del rango de los filtros existen en el sistema

Descripción de los variables	Formulário
Repetición	50
Tiempo de Gestionar un filtro (F)	T_{add}+T_{remove}
Tiempo de procesar notificaciones (N)	T_{100000}/100000
Tiempo de diferencia de gestionar un filtro (D_f)	$F_{con\ cobertura}$- $F_{sin\ cobertura}$
Tiempo de diferencia de gestionar un filtro con 1.5 veces ganancia ($D_{1.5f}$)	1.5$F_{con\ cobertura}$- $F_{sin\ cobertura}$
Tiempo de diferencia de gestionar un filtro con 2 veces ganancia (D_{2f})	2$F_{con\ cobertura}$- $F_{sin\ cobertura}$
Tiempo de diferencia de la notificación (D_n)	$N_{sin\ cobertura}$ - $N_{con\ cobertura}$
Tiempo de diferencia de la notificación con 1.5 veces ganancia ($D_{1.5n}$)	$N_{sin\ cobertura}$ - 1.5$N_{con\ cobertura}$
Tiempo de diferencia de la notificación con 2 veces ganancia (D_{2n})	$N_{sin\ cobertura}$ - 2$N_{con\ cobertura}$

Cuadro 5.2: Tabla de parámetros de experimento

Después calculamos el ratio de tiempo que está definido como:

$$Ratio = \frac{D_f}{D_n}$$

Así que sabemos si el sistema práctico tiene un ratio mayor que el ratio calculado, la cobertura sirve para mejorar el sistema, si no el sistema no puede aprovechar la cobertura. También hemos definido el caso con 1.5 veces ganancia y 2 veces ganancia, y son :

$Ratio_{1,5} = \frac{D_{1,5f}}{D_{1,5n}}$ y $Ratio_2 = \frac{D_{2f}}{D_{2n}}$.

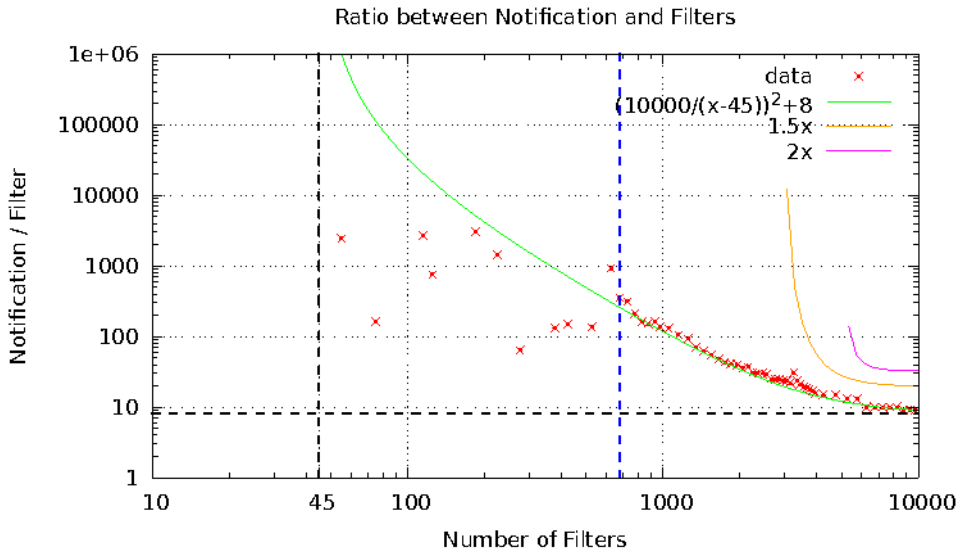

Figura 5.4: El ratio entre el filtro y la notificación

La figura 5.4 muestra resultado y las curvas ajustadas que hemos obtenido, y hay que dar cuenta

a los siguientes casos:

- **Sin cobertura** cuando con muy pocos filtros, en nuestro caso es 45, no existe la cobertura entre los filtros, por eso el sistema con cobertura y sin cobertura funciona exactamente igual en procesar los notificaciones. Y si hay diferencia en el procedimiento, es un error de medida.
- **Estimación de error** usamos los datos en caso que no exista la cobertura para calcular la estimación de error, y con eso estimamos el error relativo.
- **Filtración de datos** en primer lugar, los datos de diferencia menor que cero no se usan para calcular la curva de aproximación, además como según la estimación de error, los datos que tienen un error relativo grande tampoco no se usan en nuestro caso es de 45 hasta 625.

Los resultados están resumidos como:

- **No hay cobertura:** cuando es menor que 45 filtros, el sistema no existe la cobertura, por lo tanto, no podemos aprovechar la cobertura.
- **Con muy poca cobertura:** cuando hay pocas coberturas de 45 hasta 1000, los datos tienen error relativo grande y no son fiables, así que no podemos identificar si realmente podemos mejorar el sistema con la cobertura, pero con la curva de aproximación podemos ver que el ratio es mayor que 100 y se sube hasta infinito.
- **La cobertura está cambiando:** en este caso, el ratio está cambiando desde 100 hasta 20 según el cambio de la cobertura. Notamos que la diferencia es más pequeña según crece los filtros. Pero el número de los filtros ya es mayor que 1000, por lo tanto ya no sirve para los sistemas con pocos filtros.
- **La cobertura atiende a estable:** este caso, el ratio atiende a un constate 9.
- **Los casos con más ganancias:** cuando con 1.5 veces de ganancia, la curva ya muy difícil de aproximar y los datos de menor que 3000 filtros ya no son confiables, y atiende a un constante de 20. Además, para obtener el ratio menor que 100, el número de filtro ya es mayor que 4000. En el caso de 2 veces de ganancia, el ratio atiende a un constante de 30, y los datos mayores que 5500 son confiables.

5.3. Conclusión final

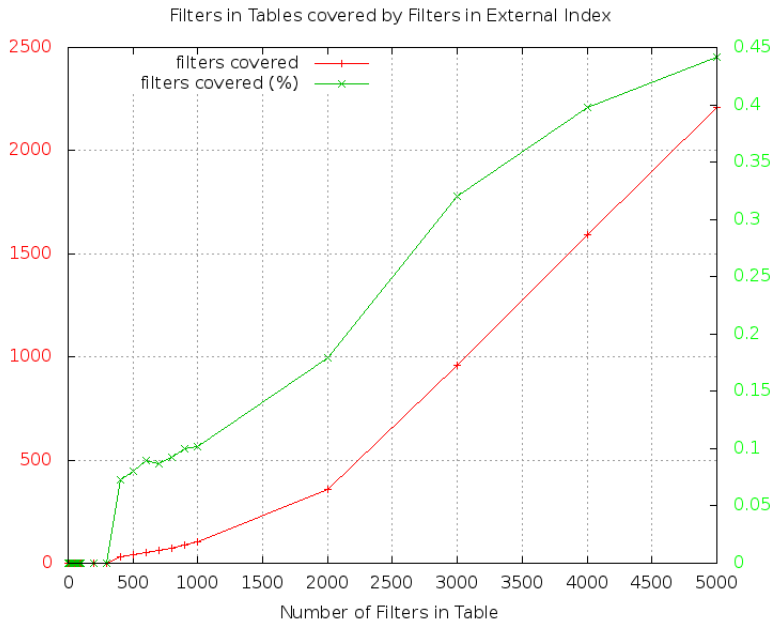

Figura 5.5: La cobertura

La figura 5.5 muestra la situación de la cobertura según el número total de filtros en el sistema, resumido los resultados anteriores con la situación de cobertura, concluimos como:

- en una visión global, la cobertura que ha producido distribución Zifp tiene una incertidumbre muy grande, y eso produce un error grande a las medidas en cuando existan pocos filtros en el sistema.
- además según los resultados básicos, podemos saber para gestionar un filtro con cobertura requiere un tiempo grande, y gana un tiempo muy pequeño en procesar una notificación con la cobertura. En particular, eso se comporta muy claro en cuando el sistema tenga muy pocos filtros con la cobertura débil.
- según el ratio podemos saber:
  - cuando el sistema obtenga más de 1000 filtros, el tiempo que perder en gestionar la cobertura se puede recuperar en procesar menor de 100 notificaciones.
  - con un número de 10000 filtros, el sistema ya está con la cobertura estable y atiende a una constante pequeña (de 10 hasta 40).
  - con menor que 100 filtros, el sistema sólo perder el tiempo en gestionar la cobertura, y no hay ganancia.

- de 100 hasta 1000, como hay un error muy grande en las medidas, resulta difícil de determinar la situación. Sin embargo, con la ayuda de los casos anteriores y las curvas de aproximaciones, podemos deducir que el ratio empieza de infinito(cuando existe la cobertura, en nuestro caso es con 45 filtros) y baja hasta 100, el cambio es muy grande, por lo tanto, resulta muy difícil de obtener un impacto bueno al sistema.

Capítulo 6

Conclusiones

En este capítulo resumimos las evaluaciones con el sistema *SilboPS* y su caso real en práctico para obtener la evaluación final. En primer lugar, voy a mencionar otra vez los propósitos de este trabajo son: seleccionar la mejor implementación de la estructura de datos y evaluar el impacto sobre el sistema general. Enseguida, voy a dar la conclusión final en los siguientes apartados.

6.1. Conclusión para la selección de las tablas de cobertura

Atendiendo a los resultados del capítulo 4, indiscutible que la tabla de conteo tiene una ventaja grande frente a la tabla general.

6.2. Conclusión para el rendimiento del sistema práctico

Desafortunadamente, el entorno concreto, que está trabajando el sistema *SilboPS*, contiene una cantidad pequeña de filtros por cada conexión (de un promedio de 16), así que según la evaluación del capítulo 5, con tan pocos filtros, no existe la cobertura entre los filtros, por lo tanto, no es posible mejorar su rendimiento con la cobertura.

6.3. Conclusión final

- La cobertura no puede mejorar un sistema como *SilboPS* que trabaja con **muy pocos filtros y con una cobertura débil o inexistente**.
- Para obtener la ganancia, el sistema que va a usar la cobertura al menos tiene que trabajar con una cantidad de filtros **mayor que 1000 por cada conexión** con nuestra implementación.

Apéndice A

Lista de palabras

1. *middleware* middleware 8
2. *publicador y suscriptor* publisher and suscriber 8
3. *enrutamiento* forwarding 8
4. *middleware sistema basado en el contenido* content-based rounting(CBR) middleware system 8
5. *algoritmo de conteo* counting algorithm 8
6. *el sistema publicador y suscriptor (Sistema Pub/Sub)* publisher and suscriber system 12
7. *algoritmo de enrutamiento* forwarding algorithm 12
8. *publicador y suscriptor software patrón* publish–subscribe messaging pattern 12
9. *broker* broker 12
10. *advertise* advertise 14
11. *reenviar* routing 14
12. *el sistema basado en el tema* topic-based system 14
13. *el sistema basado en el contenido* content-based system 14
14. *el sistema basado en el tipo* type-based system 14
15. *mensaje filtrado basado en el contenido* content-based message filtering 15
16. *el proceso de la filtración* message filtering process 15
17. *el sistema publicador y subscriptor basado en el contenido* content-based publish-suscribe system 16

18. *predicados* predicate 18
19. *filtro* filter 18
20. *cobertura* coverage 18
21. *retículo* lattice 18
22. *álgebra abstracto* abstract algebra 18
23. *conjunto parcialmente ordenado* partially ordered set 18
24. *un sistema algebraico* algebraic structures 18
25. *cubierta mínima* min cover 18
26. *límite superior* Upper Bound 18
27. *límite superior mínimo* Least Upper Bound 18
28. *límite inferior* Lower Bound 18
29. *límite inferior máximo* Great Lower Bound 18
30. *test parametrizados en JUnit* JUnit parameterized test 33
31. *Zipf* Zipf 33

Bibliografía

[1] ALEXANDER L. WOLF ANTONIO CARZANIG. Forwarding in a content based network.

[2] ALEXANDER L. WOLF ANTONIO CARZANIG, JING DENG. Fast forwarding for content-based networking.

[3] GIAN PIETRO PICCO GIANPAOLO CUGOLA. Reds: A reconfigurable dispatching system.

[4] WILLIAM J. GILBERT. *Modern Algebra with Application*.

[5] IEEE SÉRGIO DUARTE J. LEGATHEAUX MARTINS, SENIOR MMNBER. Routing algorithms for content based publish subscribe systems.

[6] NATHAN JACOBSON. *Lectures In Abstract Álgebra*, volume 1.

[7] GERARD MESZAROS. *xUnit Test Patterns*.

[8] RACHID GUERRAOUI ANNE-MARIE KERMARREC PATRICK TH. EUGSTER, PASCAL A. FELBER. The many faces of publish subscribe.

[9] WIKIPEDIA. Publish subscribe pattern.

[10] KITSANA WAIYAMAI YANEE KACHAI. Representing large concept hierarchies using lattice data structure.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Feb 14 19:34:54 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)